Final Report: 2009–2010 DARPA Computer Science Study Panel (CSSP), Phase 1
Reporting Period: April 2009–September 2010
Project: Algorithms and Software with Tunable Parallelism
PI: Prof. Richard Vuduc, School of Computational Science and Engineering at Georgia Tech

1 Costs

In the period covering Year 1 (April 2009–April 2010) plus a no-cost extension period through September 30, 2010, we spent a total of $91,714.13, broken down as follows:

- Salaries: $40,200.83
- Fringe Benefits: $4,062.75
- Tuition: $10,788.57
- Travel: $5,121.22
- Materials and supplies: $4,208.28
- Overhead: $23,740.85

2 Summary of Initial Proposal

Problem statement and goal. The goal of this project was to explore technologies that could simplify the development of high-performance computing (HPC) algorithms and software, with a specific focus on development for emerging *heterogeneous* parallel computer systems. Our motivation is to discover new techniques and tools for DoD that might help with existing HPC modernization efforts,[1] including current and future exascale[2] simulation, system acquisition, and development; massive-scale data analytics; and in-the-field HPC deployments [1].

"Heterogeneous" in this context refers to a system that contains a mix of processing components that have different computational speeds and capabilities. Our focus in this project is on a computer system comprising general-purpose multicore CPUs augmented by graphics co-processors (GPUs). The CPUs might be best suited to one type of parallel algorithm (e.g., a task-parallel one) while the GPU might be best suited to another (e.g., data-parallel). For such systems, the challenge for HPC algorithm designers and software developers is how to design and implement portable software, since different types of processing components require different strategies for design, implementation, and performance tuning.

Our approach: Tunable parallelism. In light of this problem and goal, we had proposed to develop and evaluate the concept of *tunable parallelism*. The idea is to write an algorithm and/or program in a way that simultaneously expresses many types of parallelism, with a "tuning knob" that could be automatically adjusted to fit a given heterogeneous machine.[3] To make concrete progress, we focused our efforts on developing and evaluating the concept of tunable parallelism for the following three classes of computational algorithms.

[1] See, for instance, http://www.hpcmo.hpc.mil/cms2/index.php.

[2] An exascale system is expected to be capable of performing up to $O(10^{18})$ operations per second. By contrast, the fastest current systems operate at petascales, or $O(10^{15})$ operations per second.

[3] Who or what "tunes" this knob is not constrained. For instance, this adjustment could be done by a human, by a compiler or code generator, by a runtime system, or even by the hardware itself.

1. *Dense linear algebra* (DLA). Linear algebra plays a key role in nearly all numerical algorithms. In the case of DLA, achieving high-performance was thought to be well-understood, until several years ago when researchers began observing that classical parallel DLA algorithms did not perform well on then-emerging multicore systems. Instead, these algorithms needed to be redesigned in a manner that, as it happens, is amenable to our notion of tunable parallelism [2, 3]. Our project looked for ways to *express* this tunable parallelism, which we did using a novel dataflow-based programming model called *Concurrent Collections (CnC)* [4].

2. *Generalized n-body problems* (GNPs). Analyzing massive amounts of data is a challenge of clear concern in many applications. The GNP theory is a mathematical formalism that unifies seemingly disparate statistical data analysis and machine learning tasks, such as n-point correlation, hierarchical clustering, k-nearest neighbors classification, kernel density estimation, and a variety of spatial region queries, among numerous others [5]. As the name suggests, a GNP generalizes the classical n-body problem from computational physics to this much broader class of problems in computational statistics. Our interest in GNPs stems from the ways in which we can tune data- and task-parallelism *and* trade-off numerical accuracy.

3. *Event-driven solvers* (EvDS). Many physical system simulations require numerically solving time-dependent time-dependent partial differential equations (PDEs). The traditional approach is to use so-called time-stepping methods, which are easy to parallelize but can be challenging to apply to problems in which there is highly non-uniform behavior in the solution. Turbulence is an example of such a phenomenon. To handle these problems, a few researchers recently proposed *event-driven* methods [6–10]. These can more naturally adapt to irregular solution features, but constitute a radical departure from time-stepping, because parallel event-driven methods look like parallel discrete-event simulations.[4] We hypothesized that these two approaches could be fruitfully combined with the appropriate mix of each tuned on-the-fly to match the architecture and PDE [3].

For each of the three problem classes described above (DLA, GNPs, and EvDS), we began to study the way in which one could express and exploit tunable parallelism to produce faster and more portable parallel code.

3 Summary of Research Accomplishments

Our key finding and its limitations. At a high-level, our main finding is that it is possible to express and exploit tunable parallelism, which results in highly efficient and scalable parallel code. We demonstrated this concept on real heterogeneous systems available today. However, we also encountered a number of new research challenges that need to be solved in order to make such code (a) portable to emerging platforms, (b) automatically tunable, and (c) applicable to more kinds of computations. We are tackling these challenges in follow-on projects, including our Phase 2 project. Here, we summarize our results, including their implications both for lines of inquiry and for problems of interest to DoD HPC practitioners.

[4] In particular, they have predominantly fine-grained task-parallelism in contrast to the data-parallelism that dominates time-stepping methods.

Note: Our work on this grant lead to ten publications [11–20]. Three of these papers are either finalists or winners for prizes at the major HPC conferences [11, 17, 19]. Several online HPC trade media outlets [21–24] covered a fourth paper [20].

3.1 Concurrent Collections (CnC): A new programming model for HPC

Key results [17, 18]. Concurrent Collections (CnC) is a parallel programming model under development at Intel. The CnC model allows the programmer to focus his or her attention solely on the task of expressing the computation without explicit regard for parallelism, but in a way that the available parallelism is nevertheless exposed to the execution environment, e.g., compiler, runtime system.

Our initial questions were (1) could HPC computations be expressed easily within CnC, and (2) once written in CnC, could these computations execute efficiently and scalably on real systems? Our answers to both were "yes," based on our preliminary evaluation of the approach for DLA computations. Interestingly, the model itself led to the implementation of an entirely new algorithm for the generalized symmetric dense eigenvalue problem, with our CnC implementation exceeding the best competing and highly-tuned implementation by up to 2.6×. Our work on this paper received a Best Paper Award at a major HPC conference [17].

Limitations and future directions. Although these results are encouraging, our study has three limitations. We believe these limitations can be overcome and are continuing to seek solutions for them.

First, we focused on DLA, which is a computationally *regular*. By that we mean that the available parallelism and the patterns of data access are easy to predict. A better stress test would be to consider *irregular* algorithms, that is, algorithms with unpredictable degrees of parallelism and/or complex data structures and data access patterns. We are at present investigating these cases, using GNPs and sparse linear algebra as our first test cases.

Secondly, it is not yet clear how best to express some common parallel algorithmic idioms in CnC. The most important example is a *reduction* operation, which takes a set of data items and "reduces" them to a smaller set.[5] These can be implemented in CnC but require fixing the algorithm, whereas it might be better to provide reduction as a first-class primitive in the model and allow CnC to implement the reduction freely using a different algorithm. We continue to investigate ways to provide support for this and other constructs.

Lastly, we had to *tune* our DLA implementations by hand. In short, the CnC model exposes the available parallelism but does not specify how it should be scheduled and executed. This feature of the model is its strength, but requires a sophisticated compiler and/or runtime system to exploit. In addition, the tunable parallelism concept requires additional modifications to CnC to enable merging and splitting tasks. To get our results, we worked around these limitations through manual (rather than automated) scheduling. In collaboration with Intel, we are investigating techniques to do this tuning automatically.

Implications for DoD. The main implication is that CnC is a viable model for writing multicore software, given the limitations outlined above. Indeed, it is our understanding that the DARPA Ubiquitous HPC (UHPC) team from Intel is in fact pursuing the use of

[5] For instance, an add-reduction takes a list of values and computes their sum.

CnC in their project, and we believe there is good chance of success based on our early experiences. (Note: We are not actually a participant on the Intel UHPC team.)

3.2 Algorithms and software for generalized n-body problems (GNPs)

Key results [11, 15, 16, 19]. We focused our first-year project efforts on performance analysis and tuning of parallel implementations of the so-called *fast multipole method* (FMM), which is one instance of a GNP for computational physics problems. The lessons learned here should translate to the broader class of GNPs, which form a major part of our Phase 2 project. The two main results are as follows.

First, we documented a systematic *process* for analyzing and tuning the performance of the FMM for multisocket, multicore CPU systems [15]. In short, the process has three stages. In the first stage, we treat the program and hardware as a black box, limiting analysis and tuning to simple modeling and measurement techniques. In the second stage, we assume more knowledge of the computation and machine, enabling deeper inferences and modest changes to the code. In the final stage, we assume deep knowledge of the code and machine, and therefore not only arrive at the deepest insights, but also apply the most aggressive code changes. At each stage, we showed by example what models, insights, hypotheses, and performance-enhancing optimizations might be discovered and applied.[6] Although the process we describe in our paper is actually specific to our FMM, the general outlines shed light on how to design code for other GNPs. Moreover, we believe the process could generalize to a much broader set of computations and programs.

Our second key result was to then scale-up the multicore-tuned FMM described above for use in a physical simulation of 200 million deformable red blood cells (equivalent to several drops of blood), on (a) a very large system with 200,000 processing cores,[7] and (b) a "conventional" cluster consisting of 256 GPUs.[8] This code is currently a finalist for the Gordon Bell Prize, the winner of which is to be announced in November at the 2010 ACM/IEEE Conference on Supercomputing (SC).

Limitations and future work. Though we claimed a "process" above, it is not yet clear exactly how to generalize the method to an arbitrary program, which thus remains as the main open question. To gain some additional insight, we are broadening the scope of our initial work to the entire family of GNPs (part of our Phase 2 grant), which have a roughly similar computational structure and therefore should benefit from similar techniques.

Regarding tuning, our work lays the foundations for building tunable parallel GNPs, but the actual automated mechanisms to do the tuning do not yet exist. We continue to pursue such mechanisms, for GNPs and beyond.

Implications for DoD. Massive-scale particle simulations and statistical data analysis are clear problems of interest to DoD HPC efforts. We plan to release our tuned FMM code, as well as additional portable parallel libraries for the broader family of GNPs. Moreover, we expect additional applications in data analysis for our FMM/GNP machinery. Here, we give two examples. First, we have just started exploring the use of scalable spatial

[6] These optimizations include aggressive asynchronous scheduling and reordering optimizations for explicit locality and bandwidth management.

[7] This system is the so-called Jaguar PF petaflop/s system at Oak Ridge National Laboratory.

[8] This system is the Lincoln GPU cluster at the National Center for Supercomputing Applications (NCSA) at the University of Illinois Urbana-Champaign.

queries,[9] based on GNP algorithms, for disaster planning and relief, in collaboration with Georgia Tech's Center for Geographic Information Systems (GIS). Secondly, on one of our CSSG trips we met analysts at DIA interested in efficient force-directed graph layout for visualizing social networks, which we believe is an instance of a GNP.

3.3 Tunable "fast-and-loose" synchronization

Key results [12]. We studied tuning parallelism to trade-off accuracy, through a study of how to tune the synchronization of a certain class of iterative linear solvers. Many conventional parallel algorithms assume a *bulk-synchronous* processing style. In this style, the program executes in distinct phases; within a phase, several processors may execute tasks in parallel, but between these phases, *all* processors must synchronize before proceeding to the next phase. The problem is that as the number of processing cores in a system increases, so does the cost of these bulk synchronizations.

We conducted an experimental evaluation of the effects of relaxing or aggressively avoiding synchronization altogether for a simple iterative linear solver algorithm, in particular on GPU platforms [12]. In earlier research by others, this style of algorithm is sometimes referred to as chaotic relaxation or asynchronous iteration [25, 26]. Our main finding is that we can trade-off more flops, via more iterations to converge, for a higher degree of asynchronous-parallel execution with many fewer synchronizations. These "wildly asynchronous" codes can be 1.2–2.5× faster than our best synchronized GPU implementation while achieving the same accuracy. Looking forward, this result suggests research on similarly "fast-and-loose" algorithms in the coming era of increasingly massive concurrency and relatively high synchronization or communication costs.

Limitations and future work. The main limitation of this work is that *proving* a loosely-synchronized code achieves the same answer within some tolerance is notoriously difficult [26]. Thus, it is not clear for what other kinds of computations one could perform the same trick. Still, this class of methods remains an intruiging research direction, especially for computations where there is already a relatively large uncertainty about the true solution, e.g., certain data analysis problems.

3.4 Heterogeneous architectures evaluation

Key results. As is evident from the above, a number of our results use or evaluate GPU systems [11, 13, 14, 16, 19, 27]. From this experience, we authored a position paper that analyzes the true performance gains we might reasonably expect from GPUs, compared to CPUs [20]. Previously, most reports claimed 10–100× speedups for GPUs over CPUs. Our main finding is that these reports, though not false, also do not accurately reflect the true pay-offs: most of these comparisons are frequently against *sequential and unoptimized* CPU implementations. For the computations we studied in this grant, a more reasonable pay-off from a GPU was anywhere between none to as high as 3× on current systems, when compared to well-tuned and parallelized CPU codes. Subsequent to our paper, similar independently-authored studies by other researchers have appeared, largely corroborating our findings [28, 29]. In addition, several trade news outlets have reported on our results [21–24].

[9] An example of a spatial query might be, given a geographic region, find all houses that lie within the path of a storm.

Limitations and future work. Our finding is that there is a pay-off from GPUs but that it might be smaller than expected. However, this finding is limited to the specific computations we studied, though we believe our findings will hold more broadly.

Implications for DoD. The main short-term impliciation relevant to DoD is in the area of HPC system acquisition. We believe there is potential, but investments in hardware must be accompanied by concomitant (or even higher) investments in software development and infrastructure. Such investments could be in development of libraries that can hide the complexity of GPU programming as well as programming models or compiler-based tools.

3.5 Speculative future directions: Reverse computation

Background. Toward the end of Phase 1, we began studying the third class of our target computations, EvDS, where the solution of some important time-dependent PDEs can be solved using *discrete-event* simulation techniques. In parallel discrete-event simulations, the available parallelism can be highly irregular, making discrete-event simulations among the hardest class of computations for which to achieve speedups on parallel systems.

The most aggressive form of parallelization for discrete-event simulations is based on the idea of *optimistic* (or *speculative*) execution [30]. The idea is to speculatively execute any available tasks (optimistically), and then later *rollback* tasks if it is found that they executed in an undesired order. This technique has generally been regarded as expensive in time and storage, under the assumption that rolling back a computation requires periodically saving the state of that computation and restoring it on rollback.

Key results. Prior research by others suggested a different approach to state saving, based on the idea of *reverse computation*. That is, if a task should not have executed, rather than restore the previous state, run the task "in reverse" [31]. Although this is not always possible, it can be more efficient in space and time, thereby permitting more aggressive speculative parallelization.

We began new subproject called Backstroke, which is investigating specific mechanisms to enable a compiler to produce the inverse of a function. Although this general area is not new, there has to date been no robust infrastructure for computing reverse functions. We have thus far developed a prototype infrastructure, which we are actively continuing to extend. Work on Backstroke is a collaboration with the ROSE compiler project at Lawrence Livermore National Laboratory [32].

Implications for DoD. Discrete-event simulations figure prominently in DoD war-gaming simulations, as we observed in several instances of our CSSG field trips. That a similar style of simulation can be applied to numerical solution of PDEs was, to us, a bit of a surprise. Thus, one implication is that continued investments in efficient and scalable discrete-event simulation may lead to simultaneous advances for very general parallel algorithms.

Looking more broadly, we think that reverse computation has a number of exciting applications that go beyond aggressive parallelization and discrete-event simulation. These include implementation of highly-concurrent transactional (e.g., database) systems, analyzing or debugging programs, and implementing general "undo" operations in programs.

References

[1] Peter Kogge, Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Kerry Hill, Jon Hiller, Sherman Karp, Stephen Keckler, Dean Klein, Robert Lucas, Mark Richards, Al Scarpelli, Steven Scott, Allan Snavely, Thomas Sterling, R. Stanley Williams, and Katherine Yelick. Exascale Computing Study: Technology challenges in acheiving exascale systems, September 2008. *http://users.ece.gatech.edu/~mrichard/ExascaleComputingStudyReports/ECS_reports.htm*.

[2] Alfredo Buttari, Julien Langou, Jakub Kurzak, and Jack Dongarra. A class of parallel tiled linear algebra algorithms for multicore architectures. Technical Report UT-CS-07-600 (LAPACK Working Note 191), Innovative Computing Laboratory, University of Tennessee Knoxville, September 2007. *http://www.netlib.org/lapack/lawnspdf/lawn191.pdf*.

[3] Aparna Chandramowlishwaran, Abhinav Karhu, Ketan Umare, and Richard Vuduc. Numerical algorithms with tunable parallelism. In *Proc.Wkshp. Software Tools for Multicore Systems (STMCS), at IEEE/ACM Int'l. Symp. Code Generation and Optimization (CGO)*, Boston, MA, USA, April 2008. http://people.csail.mit.edu/rabbah/conferences/08/cgo/stmcs/papers/vuduc-stmcs08.pdf.

[4] Intel®Concurrent Collections for C/C++: User's Guide, v0.3. http://software.intel.com/en-us/articles/intel-concurrent-collections-for-cc/, 2009.

[5] Alexander G. Gray and Andrew W. Moore. 'n-body' problems in statistical learning. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, Vancouver, British Columbia, Canada, December 2000.

[6] A. Lew, J.E. Marsden, M. Ortiz, and M. West. Asynchronous variational integrators. *Arch. Rational Mech. Anal.*, 2003.

[7] James J. Nutaro. *Parallel discrete event simulation with application to continuous systems*. PhD thesis, University of Arizona, 2003.

[8] Homa Karimabadi, J. Driscoll, Yuri A. Omelchenko, and N. Omidi. A new asynchronous methodology for modeling of continuous systems: Breaking the curse of the Courant condition. *J. Comp. Phys.*, 205:755–775, 2005.

[9] Kedar G. Kale and Adrian J. Lew. Parallel asynchronous variational integrators. *Int. J. Numer. Meth. Engng.*, 70:291–321, 2007.

[10] Jen-Chih Huang, Xiangmin Jiao, Richard M. Fujimoto, and Hongyuan Zha. DAG-guided parallel asynchronous variational integrators with super-elements. In *Proc. Summer Computer Simulation Conf.*, pages 691–697, 2007.

[11] Ilya Lashuk, Aparna Chandramowlishwaran, Harper Langston, Tuan-Anh Nguyen, Rahul Sampath, Aashay Shringarpure, Richard Vuduc, Lexing Ying, Denis Zorin, and George Biros. A massively parallel adaptive fast multipole method on heterogeneous architectures. In *Proc. ACM/IEEE Conf. Supercomputing (SC)*, Portland, OR, USA, November 2009. **Finalist, Best Paper**. Acceptance rate: [59/261=22.6%]. http://doi.acm.org/10.1145/1654059.1654118.

[12] Sundaresan Venkatasubramanian and Richard W. Vuduc. Tuned and wildly asynchronous stencil kernels for hybrid CPU/GPU platforms. In *Proc. ACM Int'l. Conf. Supercomputing (ICS)*, New York, NY, USA, June 2009. Acceptance rate: [47/191=25%]. http://dx.doi.org/10.1145/1542275.1542312.

[13] Nitin Arora, Ryan P. Russell, and Richard W. Vuduc. Fast sensitivity computations for numerical optimizations. In *Proc. AAS/AIAA Astrodynamics Specialist Conference*, AAS 09-435, Pittsburgh, PA, USA, August 2009. http://soliton.ae.gatech.edu/people/rrussell/FinalPublications/ConferencePapers/09AugAAS_09-392_p2pLowthrust.pdf.

[14] Nitin Arora, Aashay Shringarpure, and Richard Vuduc. Direct n-body kernels for multicore platforms. In *Proc. Int'l. Conf. Parallel Processing (ICPP)*, Vienna, Austria, September 2009. Acceptance rate: [71/220=32.3%]. http://dx.doi.org/10.1109/ICPP.2009.71.

[15] Aparna Chandramowlishwaran, Kamesh Madduri, and Richard Vuduc. Diagnosis, tuning, and redesign for multicore performance: A case study of the fast multipole method. In *Proc. ACM/IEEE Conf. Supercomputing (SC)*, New Orleans, LA, USA, November 2010. *(to appear)*.

[16] Aparna Chandramowlishwaran, Samuel Williams, Leonid Oliker, Ilya Lashuk, George Biros, and Richard Vuduc. Optimizing and tuning the fast multipole method for state-of-the-art multicore architectures. In *Proc. IEEE Int'l. Parallel and Distributed Processing Symp. (IPDPS)*, Atlanta, GA, USA, April 2010. Acceptance rate: [127/527=24.1%].

[17] Aparna Chandramowlishwaran, Kathleen Knobe, and Richard Vuduc. Performance evaluation of Concurrent Collections on high-performance multicore computing systems. In *Proc. IEEE Int'l. Parallel and Distributed Processing Symp. (IPDPS)*, Atlanta, GA, USA, April 2010. **Winner, Best Paper (software track)**. Acceptance rate: [127/527=24.1%].

[18] Aparna Chandramowlishwaran, Kathleen Knobe, and Richard Vuduc. Applying the Concurrent Collections programming model to asynchronous parallel dense linear algebra. In *Proc. ACM SIGPLAN Symp. Principles and Practice of Parallel Programming (PPoPP)*, Bangalore, India, January 2010. *(poster)*. Acceptance rate: [Papers+posters: 45/173=26.1%]. http://dx.doi.org/10.1145/1693453.1693506.

[19] Abtin Rahimian, Ilya Lashuk, Aparna Chandramowlishwaran, Dhairya Malhotra, Logan Moon, Rahul Sampath, Aashay Shringarpure, Shravan Veerapaneni, Jeffrey Vetter, Richard Vuduc, Denis Zorin, and George Biros. Petascale direct numerical simulation of blood flow on 200k cores and heterogeneous architectures. In *Proc. ACM/IEEE Conf. Supercomputing (SC)*, New Orleans, LA, USA, November 2010. *(to appear)*. **Finalist, Gordon Bell Prize**.

[20] Richard Vuduc, Aparna Chandramowlishwaran, Jee Whan Choi, Murat Efe Guney, and Aashay Shringarpure. On the limits of GPU acceleration. In *Proc. USENIX Wkshp. Hot Topics in Parallelism (HotPar)*, Berkeley, CA, USA, June 2010. Acceptance rate: [Talks: 16/68=23.5%].

[21] Tarek Chammah. Parallelism at HotPar 2010. *Real World Technologies*, July 26 2010. http://www.realworldtech.com/page.cfm?ArticleID=RWT072610001641&p=6.

[22] Michael Feldman. Postcards from the edge of parallel computing. *HPCwire*, July 30 2010.

[23] Michael Feldman. GPU computing II: Where the truth lies. *HPCwire*, June 24 2010.

[24] Miriam Boon. Feature: Inflated performance. *Int'l. Sci. Grid This Week (iSGTW)*, September 8 2010.

[25] D. Chazan and W. Miranker. Chaotic relaxation. *Linear Algebra and Its Applications*, 2(2):199–222, April 1969.

[26] Andreas Frommer and Daniel B. Szyld. On asynchronous iterations. *J. Comp. Appl. Math.*, 123(1–2):201–216, November 2000.

[27] Jee Whan Choi, Amik Singh, and Richard W. Vuduc. Model-driven autotuning of sparse matrix-vector multiply on GPUs. In *Proc. ACM SIGPLAN Symp. Principles and Practice of Parallel Programming (PPoPP)*, Bangalore, India, January 2010. Acceptance rate: [29/173=16.8%]. http://dx.doi.org/10.1145/1693453.1693471.

[28] Victor W. Lee, Changkyu Kim, Jatin Chhugani, Michael Deisher, Daehyun Kim, Anthony D. Nguyen, Nadathur Satish, Mikhail Smelyanskiy, Srinivas Chennupaty, Per Hammarlund, Ronak Singhal, and Pradeep Dubey. Debunking the 100X GPU vs. CPU myth: An evaluation of throughput computing on CPU and GPU. *ACM SIGARCH Computer Architecture News*, 38(3):451–460, June 2010. *In Proc. ACM Int'l. Symp. Comp. Arch. (ISCA)*.

[29] Rajesh Bordawekar, Uday Bondhugula, and Ravi Rao. Believe it or not! Multicore CPUs can match GPU performance for a FLOP-intensive application! In *Proc. Int'l. Conf. Parallel Architectures and Compilation Techniques (PACT)*, pages 537–538, Vienna, Austria, September 2010. *(poster)*. http://dx.doi.org/10.1145/1854273.1854340.

[30] David R. Jefferson. Virtual time. *ACM Trans. Programming Languages and Systems (TOPLAS)*, 7(3):404–425, July 1985.

[31] Christopher D. Carothers, Kalyan S. Perumalla, and Richard M. Fujimoto. Efficient optimistic parallel simulations using reverse computation. In *Proc. Wkshp. Parallel and Distributed Simulation (PADS)*, pages 126–135, Atlanta, GA, USA, 1999. http://portal.acm.org/citation.cfm?id=301467.

[32] Markus Schordan and Dan Quinlan. A source-to-source architecture for user-defined optimization. In *Proc. Euro-Par*, August 2003.